S31-61

138199

Parallel Processing Architecture for Computing Inverse Differential Kinematic Equations of the PUMA Arm

T.C. Hsia
University of California, Davis
Davis, CA 95616

← need code for corp source

G.Z. Lu and W.H. Han
Nankai University
Tiangjin, China

NA416113

1. Abstract

In advanced robot control problems, on-line computation of inverse Jacobian solution is frequently required. Parallel processing architecture is an effective way to reduce computation time. ~~In this paper~~, a parallel processing architecture is developed for the inverse Jacobian (inverse differential kinematic equation) of PUMA arm ~~[2]~~. The proposed pipeline/parallel algorithm can be implemented on IC chip using systolic linear arrays. This implementation requires 27 processing cells and 25 time units. Computation time is thus significantly reduced.

2. Introduction

In many advanced robot control problems, such as with sensor guided manipulations, it is essential that the end effector be appropriately controlled in Cartesian coordinates so that the robot can adapt to a changing environment. This means that we need to compute the inverse Jacobian in real time to provide the required differential change in joint variables for a desired differential change in position and orientation. The speed of this computation directly affects the speed of robot operation. Thus efficient algorithms for computing the inverse Jacobian are needed.

There have been efforts made recently in developing computationally efficient algorthims to solve the Jacobian problem suitable for serial computer implementation [1,2]. In addition some work has been reported in algorithm development for implementation on pipelined or parallel computer [3]. These results show that such parallel algorithms can reduce computation time significantly.

A more important requirement in robot manipulation is the computing of the inverse Jacobian solution. This is generally a troublesome problem when we try to invert the Jacobian numerically. A more direct approach is to derive an explicit solution of the inverse Jacobian for a given robot. Paul, Shimano, and Mayer [2] have shown that such solutions can be obtained by differentiating the kinematic equations. This approach has shown to result simpler inverse Jacobian solutions with regard to manipulator degeneracies and joint constraints. The inverse Jacobian of the PUMA arm has been solved specifically in [2].

In this paper, we present a pipeline/parallel algorithm and architecture for computing the PUMA arm inverse Jacobian derived in [2]. With rapid advances in VLSI technology, this type of algorithm can be readily implemented on IC chips. These special purpose chips can be connected to a host computer system to achieve real-time Cartesian space control at sufficiently high sample rate. It is noted that a study has been made recently to implement direct kinematic solution on VLSI chips to speed up computation time [4]. The goal here is to further exploit the advantages of VLSI technology for the design of customized chips dedicated to the computing of the inverse Jocobian of PUMA arm.

3. Differential Kinematic Solution of PUMA Arm

Differential changes in joint variables dq_i can be related to the different changes in translation and rotation dx, dy, dz, δ_x, δ_y, and δ_z of the end effector by the relationship

$$[d_x, d_y, d_z, \delta_x, \delta_y, \delta_z,]^T = J\,[dq_1, dq_2, \ldots, dq_n]^T \qquad (1)$$

in which n is the number of joints, and J is the Jacobian matrix. But in advanced robot control problems, we need the solution of dq_i given the desired differential change d_x, d_y, d_z, δ_x, δ_y, δ_z. That is we need to compute the inverse problem

$$[dq_1, dq_2, \ldots, dq_n]^T = J^{-1}\,[d_x, d_y, d_z, \delta x, \delta y, \delta z]^T \qquad (2)$$

This represents the inverse differential kinematic solution (inverse Jacobian) of the robot arm.

Instead of relying on the direct computing of the inverse Jacobian matrix J^{-1}, an analytical solution of the inverse Jacobian problem can be frequently obtained, and such a solution for the PUMA arm is given in [2]. For the PUMA arm, the joint variables are the six rotational joint angles $\theta_1, \theta_2, \ldots, \theta_6$. Furthermore, the differential changes in translation and rotation can be related to the differential change of the end effector homogeons matrix T [2]:

$$dT = T \cdot \Delta_T \tag{3}$$

where

$$T = \begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$dT = \begin{bmatrix} dn_x & do_x & da_x & dp_x \\ dn_y & do_y & da_y & dp_y \\ dn_z & do_z & da_z & dp_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Delta T = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore differential changes in translation and rotation can also be specified in terms of the x, y, z elements of dp, do, and da (dn vector is redundant). The desired solutions of $d\theta_i$ in terms of dp, do, and da for the PUMA arm obtained in [2] are given in the appendix. A pipeline/parallel processing architecture for computing these equations is now developed below.

4. Systolic Array Processing

VLSI technology has created a new architecture horizon in implementing parallel algorithms directly on hardware. Central to this architecture is the use of systolic linear arrays which consist of inter-connected simple and mostly identical processing cells. Algorithms that can be executed using identical operations simultaneously can take advantage of the systolic array architecture to reduce computation time.

The processing cell structure we will employ is the "inner product step processor" which performs matrix-vector multiplication using one-way pipeline algorithms. For example, computing

$$Ab = p \tag{4}$$

where A is nxm and b is mx1, can be carried out in the following recurrence manner:

$$p_i^{(o)} = 0$$

$$p_i^{(K+1)} = p_i^{(k)} + a_{ik} b_k \qquad k = 1,m, \quad i = 1,n$$

$$p_i = p_i^{(m)}$$

This operation can be implemented by a linear array of m inner product step processors shown in Figure 1.

In the following section, we will reformulate the inverse differential kinematic equation given in the appendix in terms of a set of matrix-vector multiplications which can be computed in parallel and pipelining fashion.

5. Algorithm Development

In this section, we present the matrix-vector multiplication processing schemes for computing the differentials $d\theta_i$, i = 1,2,...,6. Here we assume that the trigonometric functions required are available. Typically these functions can be generated by employing ROM look-up techniques [5,6]. The algorithm is broken down into 15 steps as described below. The notation $S_i \triangleq \sin\theta_i$ $C_i \triangleq \cos\theta_i$ are used.

(1) $A_1 b_1 = p_1$

$$A_1^T = \begin{bmatrix} dp_y & dp_x & p_x & p_y & a_y & a_x & da_x & -a_x & da_y & o_y & o_x & do_x & do_y \\ -dp_x & dp_y & p_y & -p_x & -a_x & a_y & da_y & -a_y & -da_y & -o_x & o_y & do_y & -do_x \end{bmatrix}$$

$b_1^T = [C_1\ S_1]$ $\quad p_1^T = [p_{11} \cdots p_{19}\ p_{110} \cdots p_{113}]$

output: $do_1 = p_{11}/p_{13}$

(2) $f_1 m_1 + g_1 = h_1$

$f_1^T = [p_{14}\ p_{15}\ p_{18}\ p_{110}\ -p_{111}]$ $\quad m_1 = do_1$

$g_1^T = [p_{12}\ p_{17}\ p_{14}\ p_{112}\ \ p_{113}]$

$h_1^T = [h_{11} \cdots h_{15}]$

(3) $A_2 b_2 = p_2$

$$A_2^T = \begin{bmatrix} -a_3 & d_4 & p_z & -p_{13} \\ d_4 & a_3 & p_{13} & p_z \end{bmatrix} \qquad b_2 = \begin{bmatrix} S_3 \\ C_3 \end{bmatrix}$$

$p_2^T = [p_{21} \cdots p_{24}]$

(4) $f_2 m_2 + g_2 = h_2$

$f_2^T = [C_3\ S_3\ p_{21}\ p_{23}\ p_{24}]$

$g_2^T = [a_3\ d_4\ 0 \quad 0 \quad 0]$

$m_2 = a_2,\ \ h_2^T = [h_{21} \cdots h_{25}]$

(5) $A_3 b_3 = p_3$

$A_3 = [-p_z\ p_{13}]$ $\quad b_3^T = [dp_z\ h_{11}]$ $\quad p_3 = p_{31}]$

output: $do_3 = p_{31}/h_{23}$

(6) $A_4 b_4 = p_4$

$$A_4 = \begin{bmatrix} p_z & -p_{13} \\ p_{13} & p_z \end{bmatrix} \qquad b_4 = \begin{bmatrix} h_{21} \\ h_{22} \end{bmatrix} \qquad p_4 = \begin{bmatrix} p_{41} \\ p_{42} \end{bmatrix}$$

(7) $A_5 b_5 = p_5$

$$a_5 = \begin{bmatrix} -h_{21} & h_{22} & -h_{24} \\ h_{22} & h_{21} & h_{25} \end{bmatrix} \qquad b_5^T = [dp_z\ h_{11}\ do_3]$$

$p_5^T = [p_{51}\ p_{52}]$

(8) $A_6 b_6 = p_6$

$$A_6^T = \begin{bmatrix} p_{42} & -a_z & p_{16} & -p_{51} & h_{12} & da_z & p_{111} & -o_z & h_{14} & -do_z \\ -p_{41} & -p_{16} & -a_z & -p_{52} & -da_z & h_{12} & -o_z & -p_{111} & -do_z & -h_{14} \end{bmatrix}$$

$b_6^T = [c_{23}\ s_{23}]$ $\quad p_6^T = [p_{61} \cdots p_{610}]$

outputs: $do_{23} = p_{64}/p_{61}$ $\quad do_2 = do_{23} - do_3$

(9) $f_3 m_3 + g_3 = h_3$

$f_3^T = [p_3 \ p_{63} \ p_{68} \ -p_{67}]$ $m_3 = do_{23}$

$g_3^T = [p_{65} \ p_{66} \ p_{69} \ p_{610}]$

$h_3^T = [h_{31} \cdots h_{34}]$

(10) $A_7 b_7 = p_7$

$A_7 = \begin{bmatrix} p_{15} & p_{63} \\ p_{32} & -h_{13} \end{bmatrix}$ $b_7 = \begin{bmatrix} p_{15} \\ p_{63} \end{bmatrix}$ $p_7 = \begin{bmatrix} p_{71} \\ p_{72} \end{bmatrix}$

output: $do_4 = p_{72}/p_{11}$

(11) $A_8 b_8 = p_8$

$A_8^T = \begin{bmatrix} p_{15} & p_{31} & p_{67} & p_{110} & h_{33} & -p_{67} & h_{15} \\ -p_{63} & h_{13} & p_{116} & -p_{67} & h_{15} & -p_{110} & -h_{33} \end{bmatrix}$

$b_8^T = [C_4 \ S_4]$ $p_8^T = [p_{81} \cdots p_{87}]$

(12) $f_4 m_4 + g_4 = h_4$

$f_3^T = [p_{81} \ p_{84} \ p_{86}]$ $m_4 = do_4$

$g_4^T = [p_{82} \ p_{85} \ p_{87}]$ $h_4^T = [h_{41} \ h_{42} \ h_{43}]$

(13) $A_9 b_9 = p_9$

$A_9 = \begin{bmatrix} h_{41} & -h_{32} \\ -p_{68} & p_{83} \\ -h_{42} & -h_{34} \end{bmatrix}$ $b_9 = \begin{bmatrix} c_5 \\ s_5 \end{bmatrix}$ $p_9 = \begin{bmatrix} p_{91} \\ p_{92} \\ p_{93} \end{bmatrix}$

output: $do_5 = p_{91}$

(14) $f_5 m_5 + g_5 = h_5$

$f_5 = [p_{92}]$ $g_5 = [p_{93}]$ $m_5 = do_5$ $h_5 = [h_{51}]$

(15) $A_{10} b_{10} = p_{10}$

$A_{10} = [h_{51} \ -h_{43}]$ $b_{10}^T = [c_6 \ s_6]$ $p_{10} = [p_{101}]$

output: $do_6 = p_{101}$

The data flow timing table for these computations are given in Tables 1 and 2. It is shown that the solution requires 25 time units and 27 processing cells.

The results of 25 time units is a significant reduction of computation time in comparison with that when a serial computer is used to compute the original solution. The total number of multiplications of that solution is about 150. This is equivalent to 150 time units in the systoic array processing system as opposed to the 25 time units we have achieved by exploiting parallelism.

Table 1. Data flow timing table for steps 1 through 8 which compute $d\theta_1$ $d\theta_3$ and $d\theta_3$. Numbers on top row indicate time units.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
dP_y	$-dP_x$																C_1
	dP_x	dP_y															S_1
		P_x	P_y														
			P_y	$-P_x$													
				a_y	$-a_x$												
					a_x	a_y											
						da_x	da_y										
							$-a_x$	$-a_y$									
								da_y	$-da_x$								
									o_y	o_x							
										o_x	o_y						
											do_x	do_y					
												do_y	$-do_x$				
					p_{14}												do_1, p_{12}
								p_{15}									p_{17}
										p_{18}							p_{19}
													$-p_{110}$				p_{112}
														p_{111}			p_{113}
	$-a_3$	d_4															C_3
		d_4	a_3														S_3
			p_z	p_{13}													
				$-p_{13}$	p_z												
	C_3																a_2, a_3
		S_3															d_4
			p_{21}														
					p_{23}												
						p_{24}											
					$-p_z$												dp_z
						p_{13}											h_{11}
			p_z	$-p_{13}$													h_{21}
				p_{13}	p_z												h_{22}
						h_{21}	$-h_{22}$	$-h_{24}$									dp_z
							h_{22}	h_{21}	h_{25}								h_{11}, $d\theta_3$
						p_{42}	p_{41}										C_{23}
							$-a_z$	$-p_{16}$									S_{23}
								p_{16}	$-a_z$								
									$-p_{51}$	$-p_{52}$							
										h_{12}	$-da_z$						
											da_z	h_{12}					
												p_{111}	$-o_z$				
													$-o_z$	$-p_{111}$			
														h_{14}	$-do_z$		
															$-do_z$	$-h_{14}$	

Table 2. Data flow timing table for steps 9 through 15 which compute $d\theta_4$, $d\theta_5$, $d\theta_6$.

10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
			p_{64}													p_{65}
				p_{63}												p_{66}
							p_{68}									p_{69}
								$-p_{67}$								p_{610}
p_{15}	p_{63}															p_{15}
				h_{31}	$-h_{13}$											p_{63}
				p_{15}	$-p_{63}$											C_4
					h_{31}	h_{13}										S_4
						p_{67}	p_{110}									
							p_{110}	$-p_{67}$								
								h_{33}	h_{15}							
									$-p_{67}$	$-p_{110}$						
									h_{15}	$-h_{33}$						
							p_{81}									$d\theta_4$, p_{82}
										p_{36}						p_{85}
												p_{86}				p_{87}
									h_{41}	$-h_{32}$						C_5
										$-p_{GG}$	p_{83}					S_5
											$-h_{42}$	$-h_{34}$				
													p_{92}			$d\theta_5$, p_{93}
													$-h_{43}$	h_{51}		C_6, S_6

6. Conclusion

It has been demonstrated in this paper that parallel computing architecture can be developed for the inverse differential kinematic equation of the PUMA arm. By using systolic linear arrays employed in VLSI chip design, the computation can be completed with 27 processing cells in 25 time units.

The differential kinematic equation in its original form requires about 150 multiplications to compute. If one multiplication is counted as one time unit, the parallel architecture definitely provides a substantial reduction in computation time. A customized IC chip dedicated to this algorithm can be fabricated.

7. References

[1] D. E. Orin and W. W. Schrader, "Efficient Jacobian Determination for Robot Manipulators," Robotics Research, Brady and Paul, Editors, pp. 727-734, MIT Press, 1984.

[2] R. P. Paul, B. Shimano, and G. E. Mayer, "Differential Kinematic Control Equations for Simple Manipulators," IEEE Transactions on Systems, Man, and Cybernetics, Vol., SMC-11, No. 6, pp. 456-460, June 1981.

[3] D. E. Orin, H. H. Chao, K. W. Olson, and W. W. Schrader, "Pipeline/Parallel Algorithms for the Jacobian and Inverse Dynamics Computations," proceedings, 1985 IEEE International Conference on Robotics and Automation, pp. 785-789, 1986.

[4] S. S. Leung and M. A. Shanblatt, "A VLSI Chip Architecture for the Real-Time Computation of Direct Kinematics," proceedings, 1986 IEEE International Conference on Robotics and Automation, pp. 1717-1722, 1986.

[5] S. Muroga, VLSI System Design, When and How to Design Very-Large-Scale Integrated Circuits, Wiley, 1982, pp. 313-316.

[6] C. F. Ruoff, "Fast Trig Functions for Robot Control," Robotics Age in the Beginning, C. T. Helmers, ed., Hasbronck Heights, Hayden Book, 1983, pp. 73-79.

Appendix: Inverse Differential Kinematic Equations for PUMA arm

$$d\theta_1 = \frac{C_1dP_y - S_1dP_x}{C_1P_x + S_1P_y}$$

$$d\theta_3 = \frac{d_1}{a_2(d_4C_3 - a_3S_3)}$$

$$d_1 = f_{11}(p)df_{11}(p) - f_{12}(p)df_{12}(p)$$

$$f_{11}(p) = C_1p_x = S_1P_y$$

$$df_{11}(p) = -S_1p_xd\theta_1 + C_1dp_x + C_1p_yd\theta_1 + S_1dp_y$$

$$f_{12}(p) = -dp_z$$

$$d\theta_{23} = \frac{-S_{23}d\nu_2 - C_{23}d\nu_1}{C_{23}\nu_2 - S_{23}\nu_1}$$

$$\nu_1 = -\omega_2f_{11}(p) + \omega_1p_z$$

$$\nu_2 = \omega_1f_{11}(p) + \omega_2p_z$$

$$\omega_1 = a_2C_3 + a_3$$

$$\omega_2 = d_4 + a_2S_3$$

$$d\nu_1 = -\omega_2df_{11}(p) -_aA_2f_{11}(p)C_3d\theta_3 + \omega_1dP_z - a_2S_3p_zd\theta_3$$

$$d\nu_2 = \omega_1df_{11}(p) - a_2S_3f_{11}(p)d\theta_3 + \omega_2dP_z + a_2C_3p_3d\theta_3$$

$$d\theta_2 = d\theta_{23} - d\theta_3$$

$$d\theta_4 = \frac{NC_4d(NS_4) - NS_4d(NC_4)}{(NS_4)^2 + (NC_4)^2}$$

$$NC_4 = C_{23}D_{41} - S_{23}a_z$$

$$D_{41} = C_1a_x + S_1a_y$$

$$d(NS_4) = -C_1a_xd\theta_1 - S_1a_yd\theta_1 + C_1da_z - S_1da_x$$

$$NS_4 = -S_1a_x + C_1a_y$$

$$d(NC_4) = -S_{23}D_{41}d\theta_{23} + C_{23}dD_{41} - C_{23}a_zd\theta_{23} - S_{23}da_z$$

$$dD_{41} = -S_1a_xd\theta_1 + C_1da_x + C_1a_yd\theta_1 + S_1da_z$$

$$d\theta_5 = C_5dS_5 - S_5dC_5$$

$$dS_5 = -S_4NC_4d\theta_4 + C_4d(NC_4) + C_4d\theta_4NS_4 + S_4d(NS_4)$$

$$dC_5 = C_{23}d\theta_{23}D_{41} + S_{23}dD_{41} - S_{23}a_zd\theta_{23} + C_{23}da_z$$

$$S_5 = C_4NC_4 + S_4NS_4$$

$$C_5 = S_{23}D_{41} + C_{23}a_z$$

$d\theta_6 = C_6 dS_6 - S_6 dC_6$

$S_6 = -C_5 N_{61} - S_5 N_{612}$

$C_6 = -S_4 N_{611} + C_4 N_{6112}$

$dS_6 = S_5 N_{61} d\theta_5 - C_5 dN_{61} - C_5 N_{612} d\theta_5 - S_5 dN_{612}$

$dC_6 = -dS_4 N_{611} - S_4 dN_{611} + dC_4 N_{6112} + C_4 dN_{6112}$

$= -C_4 N_{611} d\theta_4 - S_4 dN_{611} + - S_4 N_{6112} d\theta_4 + C_4 dN_{6112}$

$N_{61} = C_4 N_{611} + S_4 N_{6112}$

$N_{611} = C_{23} N_{6111} - S_{23} O_z$

$N_{6112} = -S_1 O_x + C_1 O_y$

$dN_{61} = -S_4 N_{611} d\theta_4 + C_4 dN_{611} + C_4 N_{6112} d\theta_4 + S_4 dN_{612}$

$dN_{611} = -S_{23} N_{6111} d\theta_{23} + C_{23} dN_{6111} - C_{23} O_z d\theta_{23} - S_{23} dO_z$

$dN_{6112} = - C_1 O_x d\theta_1 - S_1 dO_x - S_1 O_y d\theta_1 + C_1 dO_y$

$N_{6111} = C_1 O_x + S_1 O_y$

$dN_{6111} = -S_1 O_x d\theta_1 + C_1 dO_x + C_1 O_y d\theta_1 + S_1 dO_y$

$N_{612} = -S_{23} N_{6111} - C_{23} O_z$

$dN_{612} = -C_{23} N_{6111} d\theta_{23} - S_{23} dN_{1111} + S_{23} O_z d\theta_{23} - C_{23} dO_z$

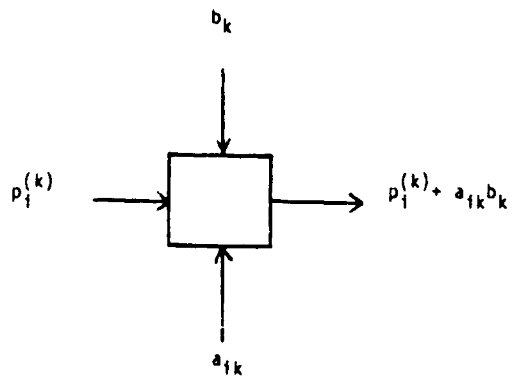

Figure 1. Inner product step processor